Verification of Restricted EA-Equivalence for Vectorial Boolean Functions

Lilya Budaghyan and Oleksandr Kazymyrov

Department of Informatics, University of Bergen,
P.O.Box 7803, N-5020 Bergen, Norway
{Lilya.Budaghyan,Oleksandr.Kazymyrov}@uib.no

Abstract. We present algorithms for solving the restricted extended affine equivalence (REA-equivalence) problem for any m-dimensional vectorial Boolean functions in n variables. The best of them has complexity $O(2^{2n+1})$ for REA-equivalence $F(x) = M_1 \cdot G(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$. The algorithms are compared with previous effective algorithms for solving the linear and the affine equivalence problem for permutations by Biryukov et. al [1].

Keywords: EA-equivalence, Matrix Representation, S-box, Vectorial Boolean Function.

1 Introduction

Vectorial Boolean functions play very important role in providing high-level security for modern ciphers. They are used in cryptography as nonlinear combining or filtering functions in the pseudo-random generators (stream ciphers) and as substitution boxes (S-boxes) providing confusion in block ciphers. Up to date an important question of generation of vectorial Boolean functions with optimal characteristics to prevent all known types of attacks remains open. Sometimes equivalence (i.e. EA or CCZ) is used for achieving necessary properties without losing other ones (i.e. δ-uniformity, nonlinearity) [2].

But very often inverse problem occurs: it is needed to check several functions for equivalence. For instance, when finding a new vectorial Boolean function it is necessary to verify whether it is equivalent to already known ones as it happens with some of block ciphers, where several substitutions are used, (i.e. ARIA [3] or Kalyna [4,5]). The complexity of exhaustive search for checking EA-equivalence for functions from $\mathbb{F}_2^n$ to itself equals $O\left(2^{3n^2+2n}\right)$. When $n = 6$ the complexity is already 2^{120} that makes it impossible to perform exhaustive computing.

In the paper [1] Alex Biryukov et al. have shown that in case when given functions are permutations of $\mathbb{F}_2^n$, the complexity of determining REA-equivalence equals $O\left(n^2 \cdot 2^n\right)$ for the case of linear equivalence and $O\left(n \cdot 2^{2n}\right)$ for affine equivalence. In this paper we consider more general cases of REA-equivalence for functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ and specify results, when complexity can be reduced to polynomial. The complexities of our algorithms and the best previous known ones are given in Table 1.

F. Özbudak and F. Rodríguez-Henríquez (Eds.): WAIFI 2012, LNCS 7369, pp. 108–118, 2012.

Table 1. Best Complexities for Solving REA-equivalence Problem

Restricted EA-equivalence	Complexity	m	$G(x)$	Source
$F(x) = M_1 \cdot G(M_2 \cdot x)$	$O\left(n^2 \cdot 2^n\right)$	$m = n$	Permutation	[1]
$F(x) = M_1 \cdot G(M_2 \cdot x \oplus V_2) \oplus V_1$	$O\left(n \cdot 2^{2n}\right)$	$m = n$	Permutation	[1]
$F(x) = M_1 \cdot G(x \oplus V_2) \oplus V_1$	$O\left(2^{2n+1}\right)$	$m \geq 1$	†	Sec. 3
$F(x) = M_1 \cdot G(x \oplus V_2) \oplus V_1$	$O\left(m \cdot 2^{3n}\right)$	$m \geq 1$	Arbitrary	Sec. 3
$F(x) = G(M_2 \cdot x \oplus V_2) \oplus V_1$	$O\left(n \cdot 2^m\right)$	$m \geq 1$	Permutation	Sec. 3
$F(x) = G(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$	$O\left(n \cdot 2^n\right)$	$m \geq 1$	Arbitrary	Sec. 3
$F(x) = M_1 \cdot G(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$	$O\left(2^{2n+1}\right)$	$m \geq 1$	‡	Sec. 3
$F(x) = M_1 \cdot G(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$	$O\left(m \cdot 2^{3n}\right)$	$m \geq 1$	Arbitrary	Sec. 3

† - G is under condition $\{2^i \mid 0 \leq i \leq m-1\} \subset \mathrm{img}(G')$ where $G'(x) = G(x) + G(0)$.
‡ - G is under condition $\{2^i \mid 0 \leq i \leq m-1\} \subset \mathrm{img}(G')$ where G' is defined as (4).

2 Preliminaries

For any positive integers n and m, a function F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ is called *differentially δ-uniform* if for every $a \in \mathbb{F}_2^n \setminus \{0\}$ and every $b \in \mathbb{F}_2^m$, the equation $F(x) + F(x+a) = b$ admits at most δ solutions [6]. Vectorial Boolean functions used as S-boxes in block ciphers must have low differential uniformity to allow high resistance to differential cryptanalysis (see [7]). In the important case when $n = m$, differentially 2-uniform functions, called *almost perfect nonlinear* (APN), are optimal (since for any function $\delta \geq 2$). The notion of APN function is closely connected to the notion of *almost bent* (AB) function [8] which can be described in terms of the *Walsh transform* of a function $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$

$$\lambda(u,v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) + u \cdot x},$$

where "$\cdot$" denotes inner products in $\mathbb{F}_2^n$ and $\mathbb{F}_2^m$, respectively. The set $\{\lambda(u,v) \mid (u,v) \in \mathbb{F}_2^n \times \mathbb{F}_2^m, v \neq 0\}$ is called the *Walsh spectrum* of F and the set $\Lambda_F = \{|\lambda(u,v)| \mid (u,v) \in \mathbb{F}_2^n \times \mathbb{F}_2^m, v \neq 0\}$ the *extended Walsh spectrum* of F. If $n = m$ and the Walsh spectrum of F equals $\{0, \pm 2^{\frac{n+1}{2}}\}$ then the function F is called AB [8]. AB functions exist for n odd only and oppose an optimum resistance to linear cryptanalysis (see [9]). Every AB function is APN but the converse is not true in general (see [10] for comprehensive survey on APN and AB functions).

The natural way of representing F as a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ is by its algebraic normal form (ANF):

$$\sum_{I \subseteq \{1,\ldots,n\}} a_I \left(\prod_{i \in I} x_i \right), \qquad a_I \in \mathbb{F}_2^m,$$

(the sum being calculated in $\mathbb{F}_2^m$). The algebraic degree $deg(F)$ of F is the degree of its ANF. F is called affine if it has algebraic degree at most 1 and it is called linear if it is affine and $F(0) = 0$.

Any affine function $A : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ can be represented in matrix form

$$A(x) = M \cdot x \oplus C, \quad (1)$$

where M is an $m \times n$ matrix and $C \in \mathbb{F}_2^m$. All operations are performed in $\mathbb{F}_2$, thus (1) can be rewritten as

$$\begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{m-1} \end{pmatrix}_x = \begin{pmatrix} k_{0,0} & \cdots & k_{0,n-1} \\ k_{1,0} & \cdots & k_{1,n-1} \\ \vdots & \ddots & \vdots \\ k_{m-1,0} & \cdots & k_{m-1,n-1} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ \cdots \\ x_{n-1} \end{pmatrix} \oplus \begin{pmatrix} c_0 \\ c_1 \\ \cdots \\ c_{m-1} \end{pmatrix}$$

with $a_i, x_i, c_i, k_{j,s} \in \mathbb{F}_2$.

Two functions $F, G : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ are called *extended affine equivalent* (EA-equivalent) if there exist an affine permutation A_1 of $\mathbb{F}_2^m$, an affine permutation A_2 of $\mathbb{F}_2^n$ and a linear function L_3 from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ such that

$$F(x) = A_1 \circ G \circ A_2(x) + L_3(x).$$

Clearly A_1 and A_2 can be presented as $A_1(x) = L_1(x)+c_1$ and $A_2(x) = L_2(x)+c_2$ for some linear permutations L_1 and L_2 and some $c_1 \in \mathbb{F}_2^m$, $c_2 \in \mathbb{F}_2^n$.

Definition 1. *Functions F and G are called restricted EA-equivalent (REA-equivalent) if some elements of the set $\{L_1(x), L_2(x), L_3(x), c_1, c_2\}$ are in $\{0, x\}$.*

There are two special cases

- linear equivalence when $\{L_3(x), c_1, c_2\} = \{0, 0, 0\}$;
- affine equivalence when $L_3(x) = 0$.

In matrix form EA-equivalence is represented as follows

$$F(x) = M_1 \cdot G(M_2 \cdot x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$$

where elements of $\{M_1, M_2, M_3, V_1, V_2\}$ have dimensions $\{m \times m, n \times n, m \times n, m, n\}$.

We say that functions F and F' from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ are *CCZ-equivalent* if there exists an affine permutation $\mathcal{L}$ of $\mathbb{F}_2^n \times \mathbb{F}_2^m$ such that $G_F = \mathcal{L}(G_{F'})$, where $G_H = \{(x, H(x)) \mid x \in \mathbb{F}_2^n\}, H \in \{F, F'\}$. CCZ-equivalence is the most general known equivalence of functions for which differential uniformity and extended Walsh spectrum are invariants. In particular every function CCZ-equivalent to an APN (respectively, AB) function is also APN (respectively, AB). EA-equivalence is a particular case of CCZ-equivalence [11]. The algebraic degree of a function is invariant under EA-equivalence but, in general, it is not preserved by CCZ-equivalence.

3 Verification of Restricted EA-Equivalence

Special types of REA-equivalence, which are considered in this paper, are shown in Table 2.

Table 2. Special types of REA-equivalence

REA-equivalence	Type
$F(x) = M_1 \cdot G(x) \oplus V_1$	I
$F(x) = G(M_2 \cdot x \oplus V_2)$	II
$F(x) = G(x) \oplus M_3 \cdot x \oplus V_1$	III
$F(x) = M_1 \cdot G(x) \oplus M_3 \cdot x \oplus V_1$	IV

Hereinafter assume that obtaining the value $F(x)$ for any x takes one step. Pre-computed values of function $F(x), F^{-1}(x)$ and corresponding substitutions are used as input for the algorithms. Thereafter, complexity of representing functions in needed form is not taken into account, as well as memory needed for data storage. This assumptions are introduced to be able to compare complexities of algorithms of the present paper with those of [1] where the same assumptions were made.

There are $2^{m \cdot n}$ choices of linear mappings. The complexity of obtaining the $m \times n$ matrix M satisfying the equation

$$F(x) = M \cdot G(x)$$

using exhaustive search method is $O(2^n \cdot 2^{m \cdot n})$, where $O(2^{m \cdot n})$ and $O(2^n)$ are complexities of checking all matrices for all possible $x \in \mathbb{F}_2^n$. Another natural method is based on system of equations. The complexity in this case depends only on the largest of the parameters n and m. Indeed, for square matrices we can benefit from the asymptotically faster Williams method based on system of equations with complexity $O(n^{2.3727})$ [12]. Besides, for $n \leq 64$ we can use 64-bit processor instructions to bring the complexity to $O(n^2)$ because two rows (columns) can be added in 1 step. Since any system of m equations with n variables can be considered as a system of k equations with k variables where $k = \max\{n, m\}$ then the complexity of solving such a system is

$$\mu = O(k^2) \ , \tag{2}$$

which gives the complexity of finding M by this method.

Proposition 1. *Any linear function $L : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ can be converted to a matrix with the complexity $O(n)$.*

Proof. We need to find an $m \times n$ matrix M satisfying

$$L(x) = M \cdot x$$

Suppose $\text{rows}_M(i) = (m_{ij})\ ,\ \forall j \in \{0, 1, \ldots, n-1\}$ and $\text{cols}_M(j) = (m_{ij})\ ,\ \forall i \in \{0, 1, \ldots, m-1\}$ are the i-th row and the j-th column of matrix M, respectively. Each value of $x \in \{2^i \mid 0 \leq i \leq n-1\}$ is equivalent to a vector with 1 at the i-th row

$$2^0 = \begin{pmatrix} 1 \\ 0 \\ \dots \\ 0 \end{pmatrix} \quad 2^1 = \begin{pmatrix} 0 \\ 1 \\ \dots \\ 0 \end{pmatrix} \quad 2^{n-1} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 1 \end{pmatrix}.$$

Clearly, every column, except the i-th, becomes zero when multiplying the matrix M to $x = 2^i$. Hence, each column of matrix M can be computed from

$$L(2^i) = \text{cols}_M(i), \; i \in \{0, 1, \dots, n-1\}.$$

For finding all columns of M it is necessary to compute n values of $L(2^i)$, $0 \leq i \leq n-1$. Consequently the complexity of transformation is $O(n)$. □

Proposition 2. *Let $F, G : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ and $G'(x) = G(x) \oplus G(0)$. Then the complexity of checking F and G for REA-equivalence of type I equals*

- *$O(2^{n+1})$ in case when for any $i \in \{0, \dots, m-1\}$ there exists $x \in \mathbb{F}_2^n$ such that $G'(x) = 2^i$;*
- *$O(m \cdot 2^{2n})$ in case G is arbitrary.*

Proof. Let $F'(x) = F(x) \oplus F(0)$. Then REA-equivalent of type I

$$F'(x) \oplus F(0) = M_1 \cdot G'(x) \oplus M_1 \cdot G(0) \oplus V_1$$

can be rewritten in the following form

$$\begin{cases} F(0) = M_1 \cdot G(0) \oplus V_1 \\ F'(x) = M_1 \cdot G'(x) \end{cases}. \quad (3)$$

In case of $G(0) = 0$ we get $V_1 = F(0)$, but in general it's necessary first to find M_1 from equation $F'(x) = M_1 \cdot G'(x)$. If the set $\{2^i \mid 0 \leq i \leq m-1\}$ is the subset of the image set of G', then the problem of finding $m \times m$ matrix M_1 is equivalent to the problem of converting linear function to matrix form with additional testing for all x in $\mathbb{F}_2^n$. It is possible to find M_1 with the complexity $O(m)$ as was shown in Proposition 1. The complexity of finding the pre-images of G' of elements 2^i, $\forall i \in \{0, \dots, m-1\}$ equals $O(2^n)$ as well as the complexity of checking $F'(x) = M_1 \cdot G'(x)$ for given M_1. In cryptography, in most cases $2^n \gg m$, so the complexity $O(m)$ can be neglected. Therefore the total complexity of verification for equivalence of F and G equals $O(2^n + 2^n + m) \approx O(2^{n+1})$.

Let now G be arbitrary and $F'(x)_i$ be the i-th bit of $F'(x)$. Denote $\text{img}(G')$ the image set of G' and $u_{G'} = |\text{img}(G')|$ the number of elements of $\text{img}(G')$. Let also $N_{G'}$ be any subset of $\mathbb{F}_2^n$ such that $|N_{G'}| = u_{G'}$ and $|\{G'(a) | a \in N_{G'}\}| = u_{G'}$. Then to find M_1 it is necessary to solve a system below for all $i \in \{0, \dots, m-1\}$

$$F'(x_j)_i = \text{rows}_{M_1}(i) \cdot G'(x_j), \;\; \forall x_j \in N_{G'}, \; 0 \leq j \leq u_{G'} - 1 \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} F'(x_0)_i = \text{rows}_{M_1}(i) \cdot G'(x_0) \\ F'(x_1)_i = \text{rows}_{M_1}(i) \cdot G'(x_1) \\ \dots \\ F'(x_{u_{G'}-1})_i = \text{rows}_{M_1}(i) \cdot G'(x_{u_{G'}-1}) \end{cases}.$$

For every i, $i \in \{0, \ldots, m-1\}$, the complexity of solving the system highly depends on $u_{G'}$ and m and equals $O(\max\{u_{G'}, m\}^2)$ according to (2). Then the total complexity of finding M_1 for all m bits is $O(m \cdot \max\{u_{G'}, m\}^2)$. If value $u_{G'} \approx 2^n$, then $O(m \cdot 2^{2n})$. □

Remark 1. If it is known in advance that functions F and G in Proposition 2 are REA-equivalent of type I, then the complexity of verification $F'(x) = M_1 \cdot G'(x)$ can be ignored and the total complexity for the case $\{2^i \mid 0 \le i \le m-1\} \subset \mathrm{img}(G')$ becomes $O(2^n)$.

Proposition 3. *Let $F, G : F_2^n \mapsto F_2^n$ and G be a permutation. Then the complexity of checking F and G for REA-equivalence of type II is $O(n)$.*

Proof. Denote $H(x) = G^{-1}(F(x))$. Then the equality $F(x) = G(M_2 \cdot x \oplus V_2)$ becomes

$$H(x) = M_2 \cdot x \oplus V_2 \; .$$

Taking $x = 0$ we get $V_2 = H(0)$ and the equivalence can be represented as $H'(x) = M_2 \cdot x$, where $H'(x) = H(x) \oplus H(0)$. The method and the complexity of finding n by n matrix M_2 are similar to finding the matrix corresponding to the linear function. Therefore, the complexity equals $O(n)$. □

Proposition 4. *Let $F, G : F_2^n \mapsto F_2^m$. Then the complexity of checking F and G for REA-equivalence of type III equals $O(n)$.*

Proof. Denote $H(x) = F(x) \oplus G(x)$, then REA-equivalence

$$F(x) = G(x) \oplus M_3 \cdot x \oplus V_1$$

takes the form

$$H(x) = M_3 \cdot x \oplus V_1 \; .$$

And we have the same situation as in Proposition 3, but with $m \times n$ matrix. Thus the complexity of finding M_3 and V_1 (or showing its non-existence) equals $O(n)$. □

Every vectorial Boolean function admits the form

$$H(x) = H'(x) \oplus L_H(x) \oplus H(0) \; , \tag{4}$$

where L_H is a linear function and H' has terms of algebraic degree at least 2.

Proposition 5. *Let $F, G : F_2^n \mapsto F_2^m$ and G' be defined by (4) for G. Then the complexity of checking F and G for REA-equivalence of type IV equals*

- $O(2^{n+1})$ *in case* $\{2^i \mid 0 \le i \le m-1\} \subset img(G')$,
- $O(m \cdot 2^{2n})$ *in case G is arbitrary.*

Algorithm 1. Checking Functions for REA-equivalence of Type IV

```
Input: F'(x), L_F(x), F(0), G'(x), L_G(x), G(0)
Output: True if F is EA-equivalent to G
for V_2 = 0 to 2^n do
   H'(x) ← G'(x ⊕ V_2);
   L_H(x) ← L_G(x ⊕ V_2);
   H(0) ← G(V_2);
   for i = 0 to m − 1 do
      x ← 2^i;
      find(2^i == G(y));
      SetColumn(M_1,i,H(y));
   end for
   V_1 ← M_1 · H(0) ⊕ F(0);
   for i = 0 to n − 1 do
      x ← 2^i;
      SetColumn(M_3,i,L_F(x) ⊕ M_1 · L_H(x));
   end for
   for i = 0 to 2^n − 1 do
      if  F(x) != M_1 · H (x ⊕ V_2) ⊕ M_3 · x ⊕ V_1  then
         goto next V_2;
      end if
   end for
   return  True
end for
return  False
```

Proof. Using (4) REA-equivalence of type IV can be rewritten as

$$F'(x) \oplus L_F(x) \oplus F(0) = M_1 \cdot G'(x) \oplus M_1 \cdot L_G(x) \oplus M_3 \cdot x \oplus M_1 \cdot G(0) \oplus V_1$$

and gives the system of equations

$$\begin{cases} F'(x) = M_1 \cdot G'(x) \\ L_F(x) = M_1 \cdot L_G(x) \oplus M_3 \cdot x \\ F(0) = M_1 \cdot G(0) \oplus V_1 \end{cases}$$

It's easy to see that for a given M_1 one can easily compute M_3 and V_1 from the second and the third equations of the system. The first equation of the system leads to the two different cases for the function G' considered in Proposition 2. Hence, according to Proposition 2, the total complexity for finding G' equals $O(2^{n+1})$ and $O(m \cdot 2^{2n})$, respectively. It should be noted that the complexity of finding the matrix M_3 is not taken into account since $2^{n+1} \gg n$. □

If we add one of V_1, V_2 values to REA-equivalence, then the complexity will increase in 2^m or 2^n times respectively. REA-equivalance with V_1, V_2 and corresponding complexities are shown in Table 1. It should be mentioned that types I and III of REA-equivalence are particular cases of type IV. But taking into account different restrictions for the function G it is necessary to check all these types of EA-equivalence.

The presented methods of verification of REA-equivalence were checked using the free open source mathematical software system Sage [13]. An example of a program for the most general case (type IV) of REA-equivalence in case $\{2^i \mid 0 \leq i \leq m-1\} \subset \mathrm{img}(G')$ is shown in Appendix A. The corresponding algorithm is presented in Algorithm 1.

4 Conclusions

The present paper studies complexities of checking functions for special cases of EA-equivalence and it is shown that for some of this cases the complexity of checking takes polynomial time. Obtained results give a practical method for checking functions on equivalence. The best result is with the complexity $O(2^{2n+1})$ for checking REA-equivalence of the form $F(x) = M_1 \cdot G\,(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$ under some condition on G.

References

1. Biryukov, A., De Canniere, C., Braeken, A., Preneel, B.: A Toolbox for Cryptanalysis: Linear and Affine Equivalence Algorithms. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 33–50. Springer, Heidelberg (2003)
2. Daemen, J., Rijmen, V.: The Design of Rijndael. Springer, Heidelberg (2002)
3. Kwon, D.: New Block Cipher: ARIA. In: Lim, J.-I., Lee, D.-H. (eds.) ICISC 2003. LNCS, vol. 2971, pp. 432–445. Springer, Heidelberg (2004)
4. Oliynykov, R., Gorbenko, I., Dolgov, V., Ruzhentsev, V.: Symmetric block cipher "Kalyna". Applied Radio Electronics 6, 46–63 (2007) (in Ukrainian)
5. Oliynykov, R., Gorbenko, I., Dolgov, V., Ruzhentsev, V.: Results of Ukrainian National Public Cryptographic Competition. Tatra Mt. Math. Publ. 47, 99–113 (2010), `http://www.sav.sk/journals/uploads/0317154006ogdr.pdf`
6. Nyberg, K.: Differentially Uniform Mappings for Cryptography. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 55–64. Springer, Heidelberg (1994)
7. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. Journal of Cryptology 4(1), 3–72 (1991)
8. Chabaud, F., Vaudenay, S.: Links between Differential and Linear Cryptanalysis. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 356–365. Springer, Heidelberg (1995)
9. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
10. Carlet, C.: Vectorial Boolean Functions for Cryptography. In: Crama, Y., Hammer, P. (eds.) Chapter of the Monography Boolean Models and Methods in Mathematics, Computer Science, and Engineering, pp. 398–469. Cambridge University Press (2010)
11. Carlet, C., Charpin, P., Zinoviev, V.: Codes, bent functions and permutations suitable for DES-like cryptosystems. Designs, Codes and Cryptography 15(2), 125–156 (1998)
12. Williams, V.V.: Breaking the Coppersmith-Winograd barrier (November 2011), `http://www.cs.berkeley.edu/~virgi/matrixmult.pdf`
13. Stein, W.A., et al.: Sage Mathematics Software (Version 4.8.2), The Sage Development Team (2012), `http://www.sagemath.org`

A Source Code for Verification of REA-equivalence of Type IV

```
#!/usr/bin/env sage

# Global variables
bits=0
length=0
k=0
P=0

def check_rEA4(F,G):
  r'''
    Return True if
      - F(x) = M1 * G(x) + M3 * x + V
      - G'(x) is permutation, where G(x) = G'(x) + L_G(x) + G(0)
  '''
  M1 = matrix(GF(2),nrows=bits,ncols=bits)
  M3 = matrix(GF(2),nrows=bits,ncols=bits)

  polF = F
  polG = G

  V1 = polF.constant_coefficient()
  V2 = polG.constant_coefficient()

  polF += V1
  polG += V2

  V1 = V1.integer_representation()
  V2 = V2.integer_representation()

  polFc=polF.coeffs()
  polFc += [P("0") for i in xrange(length-len(polFc))]
  polGc=polG.coeffs()
  polGc += [P("0") for i in xrange(length-len(polGc))]

  L1 = zero_vector(length).list()
  L2 = zero_vector(length).list()

  for i in xrange(bits):
    if polFc[1<<i] != 0:
      L1[1<<i] = polFc[1<<i]
      polFc[1<<i] = 0

    if polGc[1<<i] != 0:
      L2[1<<i] = polGc[1<<i]
      polGc[1<<i] = 0

  L1 = P(L1)
  L2 = P(L2)
  polF = P(polFc)
  polG = P(polGc)

  sboxF = range(length)
  sboxG = range(length)

  sboxL1 = [L1.subs(k(ZZ(i).digits(2))).integer_representation()
      for i in xrange(length)]
  sboxL2 = [L2.subs(k(ZZ(i).digits(2))).integer_representation()
      for i in xrange(length)]
  sboxF  = [polF.subs(k(ZZ(i).digits(2))).integer_representation
      () for i in xrange(length)]
  sboxG  = [polG.subs(k(ZZ(i).digits(2))).integer_representation
      () for i in xrange(length)]

```

```
  sboxFt=sboxF[:]
  sboxGt=sboxG[:]

  if len(set(sboxG).intersection(set([2^g for g in xrange(bits)]))) != bits:
    #print ">>> sboxG hasn't all values of {0} <<<".format([2^g for g in xrange(bits)])
    return None

  for i in xrange(bits):
    x=sboxGt.index(1<<i)
    M1.set_column(i,ZZ(sboxFt[x]).digits(base=2,padto=bits))

  sboxM = range(length)

  V = ZZ((M1*vector(GF(2),ZZ(V2).digits(base=2,padto=bits))).list(),2) ^^ V1
  for i in xrange(length):
    sboxM[i] = sboxL1[i] ^^ ZZ((M1*vector(GF(2),ZZ(sboxL2[i]).digits(base=2,padto=bits))).list(),2) ^^ V

  sboxT=sboxM[:]

  V = vector(GF(2),ZZ(sboxT[0]).digits(base=2,padto=bits))

  if sboxT[0] != 0:
    sboxT = [ g^^sboxT[0] for g in sboxT ]

  for i in xrange(bits):
    x=1<<i
    M3.set_column(i,ZZ(sboxT[x]).digits(base=2,padto=bits))

  sbox = range(length)

  sF  = [F.subs(k(ZZ(i).digits(2))).integer_representation() for i in xrange(length)]
  sG  = [G.subs(k(ZZ(i).digits(2))).integer_representation() for i in xrange(length)]

  for i in xrange(length):
    sbox[i]=vector(GF(2),ZZ(sG[i]).digits(base=2,padto=bits))

    sbox[i]=M1*sbox[i]

    tx=M3*vector(GF(2),ZZ(i).digits(base=2,padto=bits))

    sbox[i]=vector(GF(2),[ZZ(sbox[i].get(j)) ^^ ZZ(tx.get(j)) ^^ ZZ(V.get(j)) for j in xrange(len(sbox[i]))])

    sbox[i]=ZZ(sbox[i].list(),2)

  if sbox == sF:
    return [M1,M3,V]
  else:
    return None

def is_EA_equivalent(F,G,functions):

  for v2 in xrange(length):
    polG=G.subs(P("x+{0}".format(k(ZZ(v2).digits(2))))).mod(P("x^{0}+x".format(length)))

    ret=check_rEA4(F,polG)

    if ret != None:
      M1=ret[0]
      M3=ret[1]
      V1=ret[2]
```

```
      V2=vector(GF(2),ZZ(v2).digits(base=2,padto=bits))
      if functions == True:
        return [M1,V1,V2,M3]
      else:
        return True

  return False

def main(argv=None):
  global bits,length,k,P

  bits=6
  length=1<<bits
  k=GF(2^bits,'a')
  P=PolynomialRing(k,'x')

  F=P.random_element(length-1)
  G=P.random_element(length-1)

  # Test polynomials for bits=6
  #G=P("a^63*x^0 + a^61*x^1 + a^23*x^2 + a^39*x^3 + a^15*x^4 + a
      ^21*x^5 + a^57*x^6 + a^37*x^7 + a^3*x^8 + a^23*x^9 + a^26*x
      ^10 + a^40*x^11 + a^48*x^12 + a^26*x^13 + a^51*x^14 + a^43*
      x^15 + a^32*x^16 + a^13*x^17 + a^33*x^18 + a^48*x^19 + a
      ^36*x^20 + a^1*x^21 + a^11*x^22 + a^40*x^23 + a^42*x^24 + a
      ^62*x^25 + a^11*x^26 + a^22*x^27 + a^5*x^28 + a^6*x^29 + a
      ^59*x^30 + a^10*x^31 + a^51*x^32 + a^4*x^33 + a^13*x^34 + a
      ^63*x^35 + a^54*x^36 + a^26*x^37 + a^58*x^38 + a^39*x^39 +
      a^53*x^40 + a^34*x^41 + a^28*x^42 + a^27*x^43 + a^40*x^44 +
       a^25*x^45 + a^10*x^46 + a^58*x^47 + a^30*x^48 + a^34*x^49
      + a^35*x^50 + a^49*x^51 + a^53*x^52 + a^35*x^53 + a^49*x^54
       + a^7*x^55 + a^55*x^56 + a^39*x^57 + a^53*x^58 + a^29*x^59
       + a^52*x^60 + a^45*x^61 + a^9*x^62 + a^26*x^63")
  #F=P("a^44*x^0 + a^34*x^1 + a^7*x^2 + a^5*x^3 + a^51*x^4 + a
      ^40*x^5 + a^27*x^6 + a^23*x^7 + a^28*x^8 + a^63*x^9 + a^20*
      x^10 + a^38*x^11 + a^12*x^12 + a^16*x^13 + a^18*x^14 + a
      ^39*x^16 + a^53*x^17 + a^62*x^18 + a^17*x^19 + a^50*x^20 +
      a^13*x^21 + a^15*x^22 + a^29*x^23 + a^33*x^24 + a^12*x^25 +
       a^22*x^26 + a^49*x^27 + a^7*x^28 + a^43*x^29 + a^28*x^30 +
       a^53*x^31 + a^5*x^32 + a^59*x^33 + a^22*x^34 + a^26*x^35 +
       a^45*x^36 + a^39*x^37 + a^49*x^38 + a^9*x^39 + a^58*x^40 +
       a^13*x^41 + a^14*x^42 + a^43*x^43 + a^61*x^44 + a^38*x^45
      + a^10*x^46 + a^9*x^47 + a^25*x^48 + a^44*x^49 + a^30*x^50
      + a^12*x^51 + a^16*x^52 + a^24*x^53 + a^56*x^54 + a^3*x^55
      + a^40*x^56 + a^23*x^57 + a^49*x^58 + a^39*x^59 + a^58*x^60
       + a^11*x^61 + a^55*x^62 + a^29*x^63")

  print "F\t= {0}".format(F)
  print "G\t= {0}".format(G)

  ret=is_EA_equivalent(F,G,functions=True)

  if ret != False:
    [M1,V1,V2,M3]=ret
    print "EA\t\t\t\t= {0}".format(True)
    print "V1:\n{0}".format(V1)
    print "V2:\n{0}".format(V2)
    print "M1:\n{0}".format(M1)
    print "M3:\n{0}".format(M3)
  else:
    print "EA\t\t\t\t= {0}".format(False)

if __name__ == "__main__":
  sys.exit(main())
```